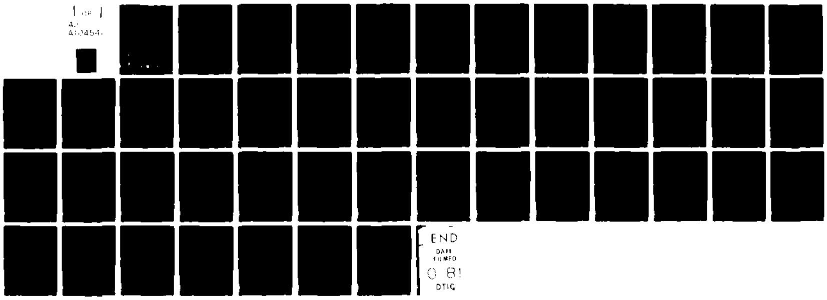
AD-A104 541 GENERAL ELECTRIC CO ARLINGTON VA INFORMATION SYSTEMS--ETC F/G 9/2
THE EFFECTS OF THE SYMBOLOGY AND SPATIAL ARRANGEMENT OF SOFTWAR--ETC(U)
AUG 81 S B SHEPPARD, J W BAILEY, E KRUESI N00014-79-C-0595
UNCLASSIFIED TR-81-388200-4 NL

1 OF 1
AD
A104541



END
DATE
FILMED
0 81
DTIC

12

AD A104541

THE EFFECTS OF THE SYMBOLOGY AND SPATIAL ARRANGEMENT OF SOFTWARE SPECIFICATIONS IN A DEBUGGING TASK

SYLVIA B. SHEPPARD

JOHN W. BAILEY

ELIZABETH KRUESI

Software Management Research
Information Systems Programs
General Electric Company
1755 Jefferson Davis Highway
Arlington, Virginia 22202

TR-81-388200-4
AUGUST 1981

GENERAL ELECTRIC

TR-81-388200-4

THE EFFECTS OF THE SYMBOLOGY AND SPATIAL ARRANGEMENT OF SOFTWARE SPECIFICATIONS IN A DEBUGGING TASK

Sylvia B. Sheppard
John W. Bailey
Elizabeth Kruesi

Software Management Research
Information Systems Programs
General Electric Company
1755 Jefferson Davis Highway
Arlington, Virginia 22202

Submitted to:

Office of Naval Research
Engineering Psychology Programs
Arlington, Virginia

Contract: N00014-79-C-0595
Work Unit: NR 196-160

August 1981

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A104 541	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) The Effects of the Symbology and Spatial Arrangement of Software Specifications in a Debugging Task.		5. TYPE OF REPORT & PERIOD COVERED (1) Technical Report.
		6. PERFORMING ORG. REPORT NUMBER (14) TR-81-388200-4
7. AUTHOR(s) Sylvia B. Sheppard, John W. Bailey, Elizabeth Kruesi		8. CONTRACT OR GRANT NUMBER(s) (15) N00014-79-C-0595
9. PERFORMING ORGANIZATION NAME AND ADDRESS Information Systems Programs General Electric Company 1755 Jefferson Davis Hwy., Arlington, VA 22202		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NR 196-160
11. CONTROLLING OFFICE NAME AND ADDRESS Engineering Psychology Programs, Code 442 Office of Naval Research Arlington, Virginia 22217		12. REPORT DATE (11) August 1981
		13. NUMBER OF PAGES 37
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) same (12) 41		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

same

18. SUPPLEMENTARY NOTES

Technical Monitor: Dr. John J. O'Hare

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Software engineering, Software experiments, Structured programming, Modern programming practices, Software documentation, Flowcharts, Program design language, Software human factors.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report describes the third in a series of experiments to evaluate the effects of the format of software specifications on programmer performance. The current experiment examined performance on a debugging task. Thirty-six professional programmers were presented with specifications for each of three modular-sized programs. Nine different specification formats were prepared for each program. These formats varied along two dimensions: type of symbology and spatial arrangement. The type of symbology included natural

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

[illegible]

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered) language, constrained language (PDL), and ideograms (flowchart symbols). The spatial arrangement included sequential (vertical flow), branching (flowchart), and hierarchical (tree-like).

The participants compared correct specifications to error-seeded program listings. Their task was to locate the several errors per program and to correct the errors using a text editor. The program output was checked automatically and a message informed the participants whether the output was correct or incorrect. The participants were asked to continue debugging until all errors had been located and corrected. The difficulty of the debugging task was measured by the time required to detect and correct the errors and by the number of submissions required for a correct run.

Substantial differences in the time to debug were associated with the type of symbology. Debugging from natural language specifications took longer than debugging from either constrained language or ideograms. This result is consistent with the results from the previous experiments in which natural language specifications were associated with longer response times in a comprehension task and in a coding task.

The overall effect of spatial arrangement was not pronounced in this experiment. However, individual combinations of symbology and spatial arrangement appeared to be differentially useful in the debugging task. Four formats resulted in a high level of performance. These were the sequential and branching constrained language versions and the branching and hierarchical ideograms.

TABLE OF CONTENTS

INTRODUCTION

Large-scale software projects necessarily involve communications among individuals with diverse skills and experience. Software design, coding, and maintenance are commonly performed by a variety of individuals at different points in time. The efficiency with which software-related tasks are performed depends critically on the documentation supplied from the previous phases of the software life cycle. The purpose of this research is to empirically evaluate a number of different documentation formats. Previous experiments in this series have examined the effects of these formats on comprehension and coding performance. The current experiment investigated performance in a debugging task.

Empirical Evaluation of Software Documentation Formats

There has been a continued interest in the relative value of flowcharts, program design language (PDL), and English prose as software development and documentation tools. An early empirical assessment of the value of flowcharts in programming was reported by Shneiderman, Mayer, McKay and Heller (1977). They performed a series of experiments on the composition, comprehension, debugging and modification of programs. For the composition task, the participants were asked to write a program; some were also asked to produce a flowchart in addition to the program. For the comprehension, debugging, and modification tasks, all participants were given a program listing while some were given a flowchart as an additional aid. Shneiderman et al. found no significant differences in any of their experiments between groups that did and did not use flowcharts.

In another study, Ramsey, Atwood, and Van Doren (1978) compared the effectiveness of flowcharts to that of a program design language. In one experiment, programmers expressed a design in either a flowchart or PDL. In a second experiment, programmers produced code from designs expressed in either a flowchart or PDL. Ramsey et al. found no difference in performance on the tasks in either experiment. However, the designs expressed in a PDL were judged to be of superior quality in that they included greater algorithmic detail, more modularization, and less abbreviation of variable names than those expressed as flowcharts.

Brooke and Duncan (1980) compared flowcharts and sequential instructions as debugging tools. They concluded that flowcharts were useful for tracing execution sequences in a program but were not helpful in conceptualizing relationships among non-contiguous segments of the program.

Although studies performed on software-related tasks have not been especially favorable to flowcharts, experiments performed in other areas of information presentation have demonstrated an advantage for flowcharts over alternative presentation formats including prose descriptions, short sentences, and decision tables (Wright and Reid, 1973; Blaiwes, 1974; Kammann, 1975). Kammann, for example, presented participants with a set of telephone dialing problems. The dialing instructions were presented in the form of a prose description or a flowchart. Fewer errors were made with the flowchart. [For a review of the non-software research, see Sheppard, Kruesi, and Curtis (1981)].

An experiment recently reported by Miller (1981) raises some doubts about the advisability of natural language as either a development or documentation tool. Miller asked non-programmers to write procedures for solving problems that were representative of common computer applications. Careful analysis of the protocols led Miller to conclude that even minor increases in the complexity of problems led to marked decreases in the quality of the solutions. Further, the high degree of contextual referencing found in the solutions provided doubts about the feasibility of adequate natural language specifications. Miller suggests that we would improve the quality of programs "...with tools that structure the problem and the implementation processes" (p.212).

Characteristics of Software Documentation

The studies described above have involved an analysis of documentation formats currently in use. A comparison of any two or more formats, such as PDL and flowcharts, may yield useful information about the relative value of these formats. This comparison does not, however, allow us to isolate the source of any observed differences since documentation formats vary along more than one dimension.

In general, there are two primary dimensions for categorizing how available documentation aids configure the information they present to programmers (Jones, 1979). The first dimension is the type of symbology in which information is presented. The second dimension is the spatial arrangement of this information. PDL, for example, uses constrained language as the symbology presented in a sequential spatial arrangement. Flowcharts use ideogram symbols presented in a branching spatial arrangement. Thus, any differences observed in the effectiveness of PDL and flowcharts may be due to the differences in the symbols, in the spatial arrangement or to an interaction of these two dimensions.

Our approach to evaluating various forms of documentation is to investigate the separate and combined effects of these two dimensions. Specifically, we have factorially combined three types of symbols with three spatial arrangements to produce nine different formats.

Type of Symbology. The symbology dimension includes natural language, constrained language, and ideograms. Documentation in the form of natural language is frequently found embedded in the source code as either global or in-line comments. Constrained language, which is embodied in a Program Design Language (PDL), is more succinct than natural language, using strictly defined keywords to describe arguments or predicates. Ideograms are frequently found in flowcharts and HIPO charts (Bohl, 1971; Katzen, 1976). A standard set of ideograms has come to represent processes or entities within a program.

Spatial Arrangement. The spatial arrangement of information in documentation is a second dimension along which documentation techniques can be categorized. In the current experiment, this dimension is represented by a sequential, a branching, and a hierarchical arrangement. A sequential arrangement is typical of narrative description, program listings and PDL while a branching arrangement is typical of flowcharts. A hierarchical arrangement is not generally used for individual module specifications but, rather, at the system level to present a visual display of the relationship among modules.

This report describes the third in a series of experiments to investigate the effects of the type of symbology and the spatial arrangement. For all experiments, the three types of symbology (natural language, constrained language, and ideograms) are factorially combined with the three spatial arrangements to produce nine different documentation formats. The first experiment, which is described in Sheppard, Kruesi, and Curtis (1981), investigated comprehension performance. The second experiment examined the influence of these dimensions on the ability of programmers to translate the specifications into code (Sheppard & Kruesi, 1981). This experiment examined the effects of these dimensions on performance in a debugging task. The results of the first two experiments are described briefly in the following sections.

Effects of Symbology and Spatial Arrangement on Comprehension

In the first experiment, seventy-two professional programmers were presented with specifications for each of three modular-sized computer programs. The participants answered a series of comprehension questions for each program using only the specifications. The questions were presented interactively on a CRT and consisted of three different types. For forward-tracing questions, the participants were given the values for a set of conditions in the program. Their task was to trace through the specifications and find the first statement executed under those conditions. For backward-tracing questions, they were required to locate a input-output questions, they were given input data and were asked to determine the value of particular variables at a later point in the program.

Both forward and backward-tracing questions were answered more quickly from specifications presented in constrained language or ideograms than in natural language. On the average, forward-tracing questions were answered most quickly from a branching arrangement and backward-tracing questions were answered more quickly from the branching and hierarchical arrangements. An examination of the individual formats revealed that the sequential constrained language (normal PDL), the branching constrained language and the branching ideogram (normal flowchart) versions were associated with very quick responses for both types of questions. For the input-output questions, no significant differences were found as a function of the type of symbology or the spatial arrangement. At the conclusion of the experimental session, participants were asked to list the type of symbology and the spatial arrangement they most preferred. Constrained language was the most preferred symbology and the branching spatial arrangement was the most preferred arrangement.

Effects of Symbology and Spatial Arrangement in a Coding Task

In the second experiment (Sheppard & Kruesi, 1981), thirty-six professional programmers were presented with specifications and partially completed code for the same three programs. The participants constructed a section of code at the middle of each program. These sections contained about fifteen lines and included the most complex decision structures present in the programs. The code was completed using a text editor, and the participants were asked to submit the program for compilation and execution. If the program did not run correctly, they were asked to correct the errors and submit it again.

Substantial differences in coding time were associated with the type of symbology. The natural language was considerably more difficult to code from than the constrained language or ideograms. An examination of the error data showed that these differences were due both to errors in coding the control flow and errors related to assignment statements and variables. The effect of the spatial arrangement was not as great as the effect of symbology. Although not statistically significant, the branching arrangement appeared to be superior to the sequential and hierarchical arrangements in minimizing control-flow errors. A comparison of the individual formats revealed that the constrained language presented in a sequential or in a branching arrangement resulted in the highest level of performance.

Again, constrained language was preferred by more participants than ideograms or natural language, and branching was the preferred spatial arrangement.

Debugging

The current experiment compared the same nine formats in a debugging task. The participants were given specifications for each of three modular-sized programs (about 50 lines of code). They compared these specifications to error-seeded program listings. Their task was to locate and correct the errors using a text editor. Performance was measured by the time required to detect and correct the errors and by the number of submissions required for a correct run.

METHOD

Participants

Thirty-six professional programmers from two different locations participated in this experiment. All were General Electric employees. The participants averaged 6.2 years of professional programming experience (S.D. = 4.9) and had used an average of 5 programming languages (S.D. = 2.3).

Independent Variables

The experiment was designed to study the effects of three independent variables: the type of symbology, the spatial arrangement of the information, and the type of program.

Program type. In our previous research (Sheppard, Curtis, Milliman & Love, 1979) significant differences in programmer performance were often associated with differences among programs. Three programs of varying types were chosen for use in this experiment. (These three programs were used in the first two experiments as well.) A program which calculated the trajectory of a rocket was chosen as representative of an engineering algorithm. An inventory system for a grocery distribution center represented the class of programs that manipulate data bases. A third program combined these two types of applications. This program interrogated a data base for information concerning the traffic pattern at an airport and simulated future needs using a queuing algorithm.

These three programs were based on algorithms contained in Barrodale, Roberts, and Ehle (1971). The algorithms were modified to incorporate only the constructs of sequence, structured iteration, and structured selection. They were then coded in Fortran and verified for correctness. Each of the resulting programs contained approximately 50 lines of executable code. In addition a short algorithm (18 lines) to find the largest of three integers was used as a practice program.

The practice program was modified to contain one error. The experimental programs each contained three errors. The errors were selected from among errors made in the coding experiment, which had used the same experimental materials. The errors included both transfer of control and assignment/variable errors but did not include syntax errors. Listings of the incorrect programs are shown in Appendix A. Handwritten corrections are included for the reader's benefit.

Type of Symbology. The statements from each program were translated into detailed specifications. Three types of symbology were used: natural language, constrained language, and ideograms. A consistent set of rules was used to map assignment, selection, and iteration statements across the three types of symbology.

Spatial Arrangements. Three spatial arrangements were used to represent the program structure: sequential, branching, and hierarchical. These three arrangements differed in the representation of control flow and nesting levels. In the sequential arrangement, both the control flow and the levels of nesting were represented vertically. In the branching arrangement, the flow of control was represented vertically while nesting levels were represented horizontally. Finally, in the hierarchical arrangement, the flow of control was represented horizontally while nesting levels were represented vertically.

Each of the three types of symbology was presented in the three spatial arrangements, resulting in nine specification formats for each program. Examples of the nine forms for the rocket trajectory program may be found in the first technical report of this series (Sheppard, Kruesi, and Curtis, 1980).

Procedure

Prior to the experiment, the participants were given a 20-minute training session in which they were shown each spatial arrangement and each type of symbology. The experimenter described the control flow for each arrangement using a sorting program as an example; this program was not seen in the actual experiment. The procedure for using the text editor to correct the programs was also explained in detail during the training session.

Experimental sessions were conducted at CRT terminals on a VAX 11/780. All coding was done in Fortran. The participants were first given a practice program containing a single error. Identical listings of the code appeared on the CRT screen and on a paper printout. The participants were told there was one error and were asked to correct the code, using the text editor. When satisfied that the program would perform correctly, a participant exited from the editor and activated a command file to compile and run the program. If the compilation was unsuccessful, a compiler message appeared on the screen directly below the line or lines containing the error. If the program compiled without errors, it was automatically executed with test data, and the output from the program appeared on the screen with one of the following messages: "OUTPUT IS CORRECT" or "OUTPUT IS INCORRECT." In the latter case, the participant was asked to keep trying until the program was correct.

Following the practice program, the three experimental programs were presented. For each program, the participants received a correct version of the specifications; these were contained on a single piece of paper. In addition, they received identical listings of the error-seeded code on the CRT screen and on a paper printout. They also received a data dictionary listing each variable, a natural language description of it, and its data type.

The participants were told that there were several errors in each experimental program and that all of them were located in the center section of the code, labeled the "COMPUTATION" section (See Appendix A). They were instructed to compare the specifications to the code, locate the errors and correct them. If a participant tried running the program without making any changes, the program compiled successfully but produced the message that the output was incorrect.

An interactive data collection system prompted the participant throughout the experimental procedure. The system recorded each change made to a program. An interval timer, accurate to the nearest second, recorded the time for each action. When a participant required more than one editing session to locate and correct the errors, the experimental system recorded exits from the editor, any compilation errors, and the incorrect outputs generated. From these data, the time to debug the programs was calculated by summing the times from the individual editing sessions; time for compiling and running the programs was not included.

On the average, the participants spent approximately 16 minutes on each experimental program. They were required to continue working on a program until all errors had been located and corrected. They were allowed to take breaks between programs.

Following the experiment, the participants completed a questionnaire about their previous programming experience. The information requested included number of years of professional experience, number of programming languages known, and whether they had previously worked with algorithms of the types used in the experiment. The participants were also asked about their preferences for type of symbology and spatial arrangement.

Design

The three types of symbology (natural language, constrained language, and ideograms) were factorially combined with the three spatial arrangements (sequential, branching, and hierarchical) to produce nine specification formats. These nine formats were constructed for each of the three programs, resulting in a total of 27 conditions.

Participants received a set of specifications for each program. Across the three programs, they saw each type of symbology and each spatial arrangement. The first participant, for example, saw the rocket trajectory program presented in sequential natural language, the inventory control program in hierarchical constrained language, and the airport traffic program in branching ideograms. The participants were assigned to conditions according to the procedures outlined in Winer (1971). [See also Kirk (1968)]. Each of the 27 conditions was used once within a set of nine participants. For this 3^3 randomized block design, a minimum of 36 participants is required to assess all interactions and main effects. Across the 36 participants, each program, symbology, and arrangement was presented first, second, and third an equal number of times.

RESULTS

Debugging Task

The participants required an average of 16 minutes to debug a program. This represents the amount of time spent studying the program and using the text editor (i.e., the total time spent at the terminal less the time for compiling, linking and running).

There were no differences among the times to debug the three programs. The rocket program required an average of 15.7 minutes, the airport program 15.8 minutes and the inventory program 16.0 minutes.

There was a significant difference among the types of symbology. The natural language versions required 18.7 minutes as compared to 14.5 minutes for the constrained language and 14.2 minutes for the ideograms (Table 1). This difference was verified by an analysis of variance ($p < .05$) (See Table 2). For this analysis, a logarithmic transformation was carried out on the times to attenuate the influence of extreme scores and to produce a more normal distribution (Kirk, 1968).

Table 1. Time to Debug (Minutes)

SPATIAL ARRANGEMENT	TYPE OF SYMBOLOGY			TOTAL
	NATURAL LANGUAGE	CONSTRAINED LANGUAGE	IDEOGRAMS	
SEQUENTIAL	19.8	12.1	18.2	16.7
BRANCHING	18.2	14.6	14.6	15.8
HIERARCHICAL	18.1	16.7	9.8	14.9
TOTAL	18.7	14.5	14.2	15.8

Note: Individual cell means represent 12 participants.

Table 2. Summary of ANOVA
Time to Debug

SOURCE	df	SS	MS	F	p
TOTAL	107	3.61			
BETWEEN PARTICIPANTS AND REPLICATIONS					
REPLICATIONS	3	.15			
PARTICIPANTS WITHIN REPLICATIONS	32	.02			
WITHIN PARTICIPANTS AND REPLICATIONS					
PROGRAM (P)	2	.01	.01	.20	
SYMBOLOGY (S)	2	.37	.18	3.60	.05
ARRANGEMENT (A)	2	.03	.01	.20	
P × S	4	.17	.04	.80	
P × A	4	.07	.02	.40	
S × A	4	.23	.06	1.20	
P × S × A	8	.24	.03	.60	
RESIDUAL	46	2.32	.05		

The effect of the spatial arrangement was not significant, and there were no significant interactions.

Number of Submissions

All of the errors in the programs were successfully located and corrected by all of the participants. An average of 2.0 submissions were required to run the programs correctly. As with the debugging times, there were no differences in number of submissions across programs.

Table 3 presents the number of submissions broken down by type of symbology and spatial arrangement. Unlike the debugging times, there were no significant differences for type of symbology. An analysis of variance indicated no significant main effects or interactions.

Preferences for Type of Symbology and Spatial Arrangement

Across the three programs, each participant received specifications in each type of symbology and in each spatial arrangement. The questionnaire indicated which three of the nine versions they had experienced during the experiment. They were asked to state which of the three versions they preferred. Table 4 shows these preferences.

Table 3. Number of Submissions Required to Complete Task

SPATIAL ARRANGEMENT	TYPE OF SYMBOLOGY			TOTAL
	NATURAL LANGUAGE	CONSTRAINED LANGUAGE	IDEOGRAMS	
SEQUENTIAL	1.8	1.9	2.5	2.1
BRANCHING	2.2	1.9	1.8	2.0
HIERARCHICAL	1.6	1.8	1.3	1.8
TOTAL	1.9	1.9	2.2	2.0

Note: Individual cell means represent 12 participants.

Table 4. Percent of Preferences for Symbology and Spatial Arrangement

SPATIAL ARRANGEMENT	TYPE OF SYMBOLOGY			TOTAL
	NATURAL LANGUAGE	CONSTRAINED LANGUAGE	IDEOGRAMS	
SEQUENTIAL	9	9	6	24
BRANCHING	15	18	25	58
HIERARCHICAL	9	6	3	18
TOTAL	33	33	34	100

The three types of symbology were preferred equally often. In terms of the spatial arrangement, branching was the most preferred, sequential was intermediate and hierarchical was the least preferred.

Experiential Factors

The questionnaire also asked for the number of years the participants had programmed professionally and the number of programming languages they had used. No correlation was found between years of experience and time to debug. Number of languages and debugging time were correlated -.26 ($p < .06$), indicating that programmers who had experience with a greater number of programming languages performed the tasks in this experiment more quickly.

DISCUSSION

The same three programs were used in the current experiment as in the comprehension and coding experiments. In the earlier experiments, significant differences in performance were associated with these three programs. Specifically, the airport-scheduling program was considerably more difficult than the inventory-control or rocket-trajectory programs. In the current experiment, no differences were observed in performance across the three programs. One possible explanation for this equality is that the relative difficulty of the errors exactly compensated for the inherent difficulty of the programs. Thus, the errors seeded in the airport-scheduling program may have been easier errors to detect and correct than those seeded in the remaining two programs. This "balancing" explanation appears unlikely since the types of errors (transfer of control and assignment/variable) and their physical locations were similar across programs.

Another possible explanation is that debugging a program from detailed specifications which are known to be correct does not require as much knowledge of the intricacies of the algorithm as does comprehending the specifications or coding from the specifications. Thus, the inherent difficulty of the algorithm may be less important in this type of a debugging task than in the earlier comprehension and coding tasks.

Differences in the type of symbology followed the pattern established in the first two experiments: the natural language versions resulted in significantly longer response times than the constrained language and ideogram versions. Had the natural language been written casually, one could hypothesize that it was incomplete and misleading. However, the natural language was developed very precisely. Assignment, selection and iteration statements were translated from the original code into the three types of symbology according to a rigid set of rules to insure that the natural language specifications were as complete and precise as the constrained language and ideograms. It is reasonable to conclude, therefore, that the differences were due to real differences among the types of symbology rather than to an experimental artifact. When combined with identical conclusions from the two previous experiments in this series, this result presents strong evidence that detailed program specifications should be presented in a more succinct symbology than natural language.

No pronounced effect for spatial arrangement appeared in this experiment. This result agrees with results from the coding experiment, where time to code and debug showed no significant effect due to spatial arrangement.

The comprehension experiment differed from this experiment and the coding experiment in that there were differences among the spatial arrangements. Forward-tracing questions were answered most quickly from the branching arrangement, and backward-tracing questions were answered more quickly from the branching and hierarchical arrangements. Response times for input-output questions did not vary significantly as a function of spatial arrangement. One explanation for the differing results among the experiments is that programming activities relating to control flow (such as tracing) benefit from the more pictorial branching and hierarchical arrangements, while other activities are not affected by the spatial arrangement. This explanation is supported by the Brooke and Duncan results presented in the Introduction.

One interesting result found in all three experiments was that the sequential and branching constrained language versions were consistently associated with low response times and a small number of errors. In cases where another version was associated with a lower response time (e.g. the hierarchical ideogram version in this experiment), differences among the two constrained language versions and the other version were not statistically significant. Of the software specifications currently in use (i.e. natural language, PDL, and flowcharts), it appears that PDL results in faster and less error-prone performance than natural language specifications; flowcharts appear in between. Sequential PDL has the additional advantage of being easy to produce at a terminal and easy to read automatically.

The participants in this experiment had no distinct preference for any of the three types of symbology. This result was surprising because in the previous two experiments constrained language was preferred, ideograms were second and natural language was least preferred. As in the previous experiments, the branching arrangement was the most preferred, the sequential arrangement was intermediate and the hierarchical arrangement was preferred least.

Diversity of experience, in terms of the number of languages used, was a better predictor of performance than years of experience. This result replicates results from the comprehension experient and our previous research (Sheppard, Milliman & Curtis, 1979) and highlights the importance of ensuring that programmers have an opportunity to gain broad applications experience as part of their professional development.

This experiment provides additional evidence that specification format can have a significant effect on the performance of programmers on software-related tasks. A debugging task was carried out more quickly from specifications presented in a succinct symbology. An examination of the individual cell means revealed four formats that led to a high level of performance. These were the constrained language presented in a sequential and in a branching arrangement and the ideograms presented in a branching and in a hierarchical arrangement. Natural language led to consistently poor performance, regardless of the spatial arrangement.

ACKNOWLEDGEMENTS

The authors would like to thank Dave Morris and Pete McEvoy for designing the automatic data collection system; Len Johnson, Charlie Burns, Jim Coughlin, Jim Sanchack, Rob Sandler and Joe Bevacqua for providing participants and facilities; Dr. John O'Hare for advice, and Tom McDonald for preparing materials and statistical analyses.

REFERENCES

Barrodale, I., Roberts, F.D.K., & Ehle, B.L. Elementary computer applications in science, engineering, and business. New York: Wiley, 1971.

Blaiwes, A.S. Formats for presenting procedural instructions. Journal of Applied Psychology, 1974, 59, 683-686.

Bohl, M. Flowcharting techniques. Palo Alto, CA: Science Research Associates, 1971.

Brooke, J.B. & Duncan, K.D. Experimental studies of flowchart use at different stages of program debugging. Ergonomics, 1980, 23, 1057-1091.

Jones, C. A survey of programming design and specification techniques. In Proceedings of the IEEE Conference on Specifications of Reliable Software. New York: Institute of Electrical and Electronics Engineers, 1979.

Kammann, R. The comprehensibility of printed instructions and the flowchart alternative. Human Factors, 1975, 17, 183-191.

Katzen, H. Systems design and documentation: An introduction to the HIPO method. New York: Van Nostrand Reinhold, 1976.

Kirk, R.E. Experimental design procedures for the behaviorial sciences. Belmont, CA: Brooks-Cole, 1968.

Miller, L.A. Natural language programming: styles, strategies, and contrasts. IBM Systems Journal, 1981, 20, 184-215.

Ramsey, H.R., Atwood, M.E., & Van Doren, J.R. A comparative study of flowcharts and program design languages for the detailed procedural specification of computer programs. (Tech. Rep. #SAI-78-078-DEN). Denver: Science Applications, Inc. 1978.

Sheppard, S.B., Curtis, B., Milliman, P., & Love, T. Modern coding practices and programmer performance. Computer, 1979, 12, (12), 41-49.

Sheppard, S.B. & Kruesi, E. The effects of the symbology and spatial arrangement of software specifications in a coding task. In Proceedings of Trends & Applications 1981: Advances in Software Technology, IEEE, 1981.

Sheppard, S.B., Kruesi, E., & Curtis, B. The effects of symbology and spatial arrangement on the comprehension ot software specifications. (Tech. Rep. TR-80-388200-2). Arlington, VA: General Electric, Information Systems Programs, 1980.

Sheppard, S.B., Kruesi, E., & Curtis, B. The effects of symbology and spatial arrangement on the comprehension of software specifications. In _Proceedings of the Fifth International Conference on Software Engineering_, IEEE, 1981.

Sheppard, S.B., Milliman, P., & Curtis, B. _Experimental evaluation of on-line program construction_ (Tech. Rep. TR-79-388100-6). Arlington, VA: General Electric, Information Systems Programs, 1979.

Shneiderman, B., Mayer, B.R., McKay, D., & Heller, P. Experimental investigations on the utility of detailed flowcharts in programming. _Communications of the ACM_, 1977, _20_, 373-381.

Winer, B.J. _Statistical principles in experimental design_. New York: McGraw-Hill, 1971.

Wright, P. & Reid, F. Written information: Some alternatives to prose for expressing the outcomes of complex contingencies. _Journal of Applied Psychology_, 1973, _57_, 160-166.

APPENDIX A

ERROR-SEEDED PROGRAM LISTINGS

PRACTICE PROGRAM

Find the largest of three integers, I, J, & K.

```
      OPEN(UNIT=1,NAME='PRAC.DAT',TYPE='OLD')
      READ (1,60) I, J, K
      IF (I .GT. J) GO TO 20
      IF (J .GT. K) GO TO 10
      LARGE = K
      GO TO 40
10    ~~J = LARGE~~   LARGE=J
      GO TO 40
20    IF (I .GT. K) GO TO 30
      LARGE = K
      GO TO 40
30    LARGE = I
40    PRINT 70,LARGE
      CLOSE(UNIT=1)
60    FORMAT (3I3)
70    FORMAT (10X,'LARGEST =    I3)
      STOP
      END
```

ROCKET PROGRAM

```
      INTEGER MAXT, TIME, FLAG
      REAL VACCEL, VVELOC, VDIST, HACCEL, HVELOC, HDIST,
     1 ANGLE, TILT, GRAV, MASS, FUEL, FORCE
C
C
C   INITIALIZATION
C
C
      VACCEL = 0.
      VVELOC = 0.
      VDIST = 0.
      HACCEL = 0.
      HVELOC = 0.
      HDIST = 0.
      ANGLE = 0.
      TILT = 0.3491
      GRAV = 32.
      MASS = 10000.
      FUEL = 50.
      FORCE = 400000.
      MAXT = 200
      FLAG = 0
      TIME = 1
C
C
C   COMPUTATION:
C
C
   10 IF (FLAG .NE. 0) GO TO 60
      IF (TIME .~~GE~~ GT 100) GO TO 20
      MASS = MASS - FUEL
      IF (TIME .NE. 11) GO TO 30
      ANGLE = TILT
      GO TO 30
   20 IF (TIME .NE. ~~100~~ 101) GO TO 30
      FORCE = 0.0
   30 VACCEL = ((FORCE * COS(ANGLE))/MASS) - GRAV
      VVELOC = VVELOC + VACCEL
      VDIST = VDIST + ~~VACCEL~~ VVELOC
      HACCEL = (FORCE * SIN(ANGLE))/MASS
      HVELOC = HVELOC + HACCEL
      HDIST = HDIST + HVELOC
      TIME = TIME + 1
      IF (VDIST .GT. 0) GO TO 40
      FLAG = 1
   40 IF (TIME .LE. MAXT) GO TO 10
      FLAG = 2
```

```
C
C
C    TERMINATION:
C
C
   60   TIME = TIME - 1
        IF (VDIST .GT. 0) GO TO 80
   70   WRITE(6,3000) TIME, HDIST
        GO TO 90
   80   WRITE(6,4000) TIME, MASS, VACCEL, VVELOC, VDIST,
     1  HACCEL,HVELOC,HDIST
   90   CONTINUE
        STOP
 3000   FORMAT(5X,'ROCKET HIT GROUND AT TIME=',I5,'SECONDS'
     1  5X,'HORIZONTAL DIST = ',F11.2)
 4000   FORMAT(5X,'ROCKET STILL ALOFT AT TIME = ',I5,
     1  ' SECONDS'/5X,'MASS = ',F22.2/
     2  5X,'VERTICAL ACCEL = ',F12.2/
     3  5X,'VERTICAL VELOC = ',F12.2/
     4  5X,'VERTICAL DIST = ',F13.2/
     5  5X,'HORIZONTAL ACCEL = ',F10.2/
     6  5X,'HORIZONTAL VELOC = ',F10.2/
     7  5X,'HORIZONTAL DIST = ',F11.2)
        END
```

INVENTORY PROGRAM

```
      INTEGER DELIV, FLAG, ITEM, ONHAND, ORDER, RELEV,
     1 REORD, STORE, UNFILL
      REAL GTOTAL, PRICE, TOTAL
C
C
C   INITIALIZATION:
C
C
      OPEN (UNIT=1, NAME='ORDERS.DAT', TYPE='OLD')
      OPEN (UNIT=2, NAME='PURCHAS.DAT', TYPE='OLD',
     1 ACCESS='SEQUENTIAL')
C
C
C   COMPUTATION:
C
C
   10 READ (1, 100, END=80) STORE
      GTOTAL = 0
      WRITE (6, 110) STORE
   20 READ (1, 120) ITEM, ORDER
      IF (ITEM .EQ. 0) GO TO 70
      CALL FETCH2(ITEM, PRICE, ONHAND, RELEV, REORD, FLAG)
      IF (ONHAND .LE. ORDER) GO TO 30
      DELIV = ORDER
      ONHAND = ONHAND - ORDER
      UNFILL = 0  GO TO 40
   30 DELIV = ONHAND
      ONHAND = ~~ONHAND - ORDER~~  O
      UNFILL = ORDER - DELIV
   40 IF (ONHAND .GT. RELEV) GO TO 50
      IF (FLAG .EQ. 0)  FLAG = 1
   50 TOTAL = DELIV * PRICE  +
      GTOTAL = GTOTAL ~~*~~ TOTAL
      IF (FLAG .NE. 1) GO TO 60
      WRITE (2, 130) ITEM, REORD
      FLAG = 2
   60 WRITE(6,140) ITEM, PRICE, ORDER, DELIV, UNFILL,TOTAL
      CALL UPDATE (ITEM, ONHAND, FLAG)
      GO TO 20
   70 WRITE (6, 150) GTOTAL
      GO TO 10
```

```
C
C
C   TERMINATION:
C
C
   80 CLOSE (UNIT=1)
      CLOSE (UNIT=2)
      STOP
  100 FORMAT (I2)
  110 FORMAT (//, 5X, 'INVOICE FOR STORE NUMBER: ', I3)
  120 FORMAT (I3, I5)
  130 FORMAT (2I7)
  140 FORMAT (5X, 'ITEM NUMBER: ', I11 / 5X,
     1 'PRICE PER ITEM:  $', F5.2 / 5X, 'NUMBER ORDERED: ',
     2 I8, /5X, 'NUMBER DELIVERED: ', I6/ 5X,
     3 'UNABLE TO DELIVER: ', I5/5X, 'TOTAL PRICE:  $', F8.2)
  150 FORMAT (/, 5X, 'TOTAL PRICE FOR ALL ITEMS: $', F10.2)
      END
```

AIRPORT PROGRAM

```
      INTEGER  ARRQUE, BEGINT, CLEAR, DEPQUE, ENDT, MAXWT
      INTEGER  NUMARR, NUMDEP, TIME, TOLWT
      REAL  ARPROB, DPPROB, RAND1, RAND2, RSEED
C
C
C  INITIALIZATION:
C
C
      RSEED = 0.0
      NUMARR = 0
      NUMDEP = 0
      CALL FETCH1(BEGINT, ARPROB, DPPROB, ARRQUE, DEPQUE,
     1 CLEAR, TOLWT)
      TIME = BEGINT
      ENDT = BEGINT + 20
C
C
C  COMPUTATION:
C
C
   10 IF (TIME .GT. ENDT) GO TO 60
      RAND1 = RND(RSEED)
      IF (RAND1 .GT. ARPROB) GO TO 20
      ARRQUE = ARRQUE + 1
   20 RAND2 = RND(RSEED)
      IF (RAND2 .GT. DPPROB) GO TO 30
      DEPQUE = DEPQUE + 1
   30 CONTINUE
      IF (CLEAR .GT. TIME) GO TO 50
      IF (ARRQUE .~~GT~~ LE. 0) GO TO 40
      ARRQUE = ARRQUE - 1
      NUMARR = NUMARR + 1
      CLEAR = TIME + 3
      GO TO 50
   40 IF (DEPQUE .LE. 0) GO TO 50
      DEPQUE = DEPQUE ~~+~~ − 1
      NUMDEP = NUMDEP + 1
      CLEAR = TIME + 2
   50 TIME = TIME + 1
      GO TO 10
   60 MAXWT = ~~(ENDT-CLEAR)~~ CLEAR - ENDT + (ARRQUE*3) + (DEPQUE*2)
```

```
C
C
C   TERMINATION:
C
C
       WRITE (6, 100) ENDT, ARRQUE, NUMARR, DEPQUE,
     1 NUMDEP,MAXWT
       IF (MAXWT .GT. TOLWT) GO TO 70
       WRITE (6, 120)
       GO TO 80
   70  WRITE (6, 110)
   80  CONTINUE
       STOP
  100  FORMAT (6X, 'ENDING TIME FOR SIMULATION: ', I5,/,
     1 12X,'ARRIVAL QUEUE: ', I5/11X,'NUMBER ARRIVED: ',I5/
     1 10X,'DEPARTURE QUEUE: ', I5/10X,'NUMBER DEPARTED: ',
     1 I5/ 13X, 'MAXIMUM WAIT: ', I5, ' MINUTES')
  110  FORMAT (5X, 'OPEN ANOTHER RUNWAY')
  120  FORMAT (5X, 'ANOTHER RUNWAY NOT NEEDED')
       END
```

TECHNICAL REPORTS DISTRIBUTION LIST

OFFICE OF NAVAL RESEARCH

Code 442

TECHNICAL REPORTS DISTRIBUTION LIST

OSD

CDR Paul R. Chatelier
Office of the Deputy Under Secretary
of Defense
OUSDRE (E&LS)
Pentagon, Room 3D129
Washington, D.C. 20301

Department of the Navy

Engineering Psychology Programs
Code 442
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217 (5 cys)

Director
Communication & Computer Technology
Code 240
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Director
Manpower, Personnel and Training
Code 270
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Information Systems Program
Code 411-IS
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Physiology Program
Code 441
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Special Assistant for Marine
Corps Matters
Code 100M
Office of Naval Research
800 North Quincy Street
Arlington, VA 22217

Department of the Navy

Commanding Officer
ONR Eastern/Central Regional Office
ATTN: Dr. J. Lester
Building 114, Section D
666 Summer Street
Boston, MA 02210

Commanding Officer
ONR Branch Office
ATTN: Dr. C. Davis
536 South Clark Street
Chicago, IL 60605

Commanding Officer
ONR Western Regional Office
ATTN: Dr. E. Gloye
1030 East Green Street
Pasadena, CA 91106

Office of Naval Research
Scientific Liaison Group
American Embassy, Room A-407
APO San Francisco, CA 96503

Director
Naval Research Laboratory
Technical Information Division
Code 2627
Washington, D.C. 20375 (6 cys)

Dr. Robert G. Smith
Office of the Chief of Naval
Operations, OP987H
Personnel Logistics Plans
Washington, D.C. 20350

Dr. Jerry C. Lamb
Combat Control Systems
Naval Underwater Systems Center
Newport, RI 02840

Naval Training Equipment Center
ATTN: Technical Library
Orlando, FL 32813

Department of the Navy

Human Factors Department
Code N215
Naval Training Equipment Center
Orlando, FL 32813

Dr. Alfred F. Smode
Training Analysis and Evaluation Group
Naval Training Equipment Center
Code N-OOT
Orlando, FL 32813

Mr. Louis Chmura
Code 7503
Naval Research Laboratory
Washington, DC 20375

Dr. Gary Poock
Operations Research Department
Naval Postgraduate School
Monterey, CA 93940

Dean of Research Administration
Naval Postgraduate School
Monterey, CA 93940

Mr. Warren Lewis
Human Engineering Branch
Code 8231
Naval Ocean Systems Center
San Diego, CA 92152

Dr. A. L. Slafkosky
Scientific Advisor
Commandant of the Marine Corps
Code RD-1
Washington, D.C. 20380

J. B. Blankenheim
Code 47013
Naval Electronics Systems Command
NC Bldg. #1, Room 4E40
Washington, DC 20360

Commanding Officer
MCTSSA
Marine Corps Base
Camp Pendleton, CA 92055

Department of the Navy

Mr. Arnold Rubinstein
Naval Material Command
NAVMAT 0722 - Rm. 508
800 North Quincy Street
Arlington, VA 22217

Commander
Naval Air Systems Command
Human Factors Programs
NAVAIR 340F
Washington, D.C. 20361

Commander
Naval Air Systems Command
Crew Station Design,
NAVAIR 5313
Washington, D.C. 20361

Mr. Phillip Andrews
Naval Sea Systems Command
NAVSEA 0341
Washington, D.C. 20362

Commander
Naval Electronics Systems Command
Human Factors Engineering Branch
Code 4701
Washington, D.C. 20360

Mr. John Impagliazzo
Code 101
Newport Laboratory
Naval Underwater Systems Center
Newport, RI 02840

CDR Robert Biersner
Naval Medical R&D Command
Code 44
Naval Medical Center
Bethesda, MD 20014

Dr. Arthur Bachrach
Behavioral Sciences Department
Naval Medical Research Institute
Bethesda, MD 20014

Dr. George Moeller
Human Factors Engineering Branch
Submarine Medical Research Lab
Naval Submarine Base

Department of the Navy

Dr. Mel C. Moy
Code 302
Naval Personnel R&D Center
San Diego, CA 92152

Dr. Richard Neetz
Code 1226
Pacific Missile Test Center
Pt Mugu, CA 93042

Navy Personnel Research and
Development Center
Planning & Appraisal
Code 04
San Diego, CA 92152

Navy Personnel Research and
Development Center
Management Systems, Code 303
San Diego, CA 92152

Navy Personnel Research and
Development Center
Performance Measurement &
Enhancement
Code 309
San Diego, CA 92152

Dr. Julie Hopson
Human Factors Engineering Division
Naval Air Development Center
Warminster, PA 18974

Mr. Jeffrey Grossman
Human Factors Branch
Code 3152
Naval Weapons Center
China Lake, CA 93555

Human Factors Engineering Branch
Code 1226
Pacific Missile Test Center
Point Mugu, CA 93042

Mr. J. Williams
Department of Environmental
Sciences
U.S. Naval Academy
Annapolis, MD 21402

Department of the Navy

Dean of the Academic Departments
U.S. Naval Academy
Annapolis, MD 21402

Human Factors Section
Systems Engineering Test
Directorate
U.S. Naval Air Test Center
Patuxent River, MD 20670

Human Factor Engineering Branch
Naval Ship Research and Development
Center, Annapolis Division
Annapolis, MD 21402

CDR W. Moroney
Code 55MP
Naval Postgraduate School
Monterey, CA 93940

Mr. Merlin Malehorn
Office of the Chief of Naval
Operations (OP-115)
Washington, D.C. 20350

Department of the Army

Mr. J. Barber
HQS, Department of the Army
DAPE-MBR
Washington, D.C. 20310

Dr. Joseph Zeidner
Technical Director
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Director, Organizations and
Systems Research Laboratory
U.S. Army Research Institute
5001 Eisenhower Avenue
Alexandria, VA 22333

Technical Director
U.S. Army Human Engineering Labs
Aberdeen Proving Ground, MD 21005

Department of the Army

ARI Field Unit-USAREUR
ATTN: Library
C/O ODCSPER
HQ USAREUR & 7th Army
APO New York 09403

Department of the Air Force

U.S. Air Force Office of Scientific Research
Life Sciences Directorate, NL
Bolling Air Force Base
Washington, D.C. 20332

Chief, Systems Engineering Branch
Human Engineering Division
USAF AMRL/HES
Wright-Patterson AFB, OH 45433

Air University Library
Maxwell Air Force Base, AL 36112

Dr. Earl Alluisi
Chief Scientist
AFHRL/CCN
Brooks AFB, TX 78235

Foreign Addressees

North East London Polytechnic
The Charles Myers Library
Livingstone Road
Stratford
London E15 2LJ
ENGLAND

Professor Dr. Carl Graf Hoyos
Institute for Psychology
Technical University
8000 Munich
Arcisstr 21
FEDERAL REPUBLIC OF GERMANY

Dr. Kenneth Gardner
Applied Psychology Unit
Admiralty Marine Technology Establishment
Teddington, Middlesex TW11 OLN
ENGLAND

Foreign Addressees

Director, Human Factors Wing
Defence & Civil Institute of Environmental Medicine
Post Office Box 2000
Downsview, Ontario M3M 3B9
CANADA

Dr. A. D. Baddeley
Director, Applied Psychology Unit
Medical Research Council
15 Chaucer Road
Cambridge, CB2 2EF
ENGLAND

Other Government Agencies

Defense Technical Information Center
Cameron Station, Bldg. 5
Alexandria, VA 22314 (12 cys)

Dr. Craig Fields
Director, Cybernetics Technology Office
Defense Advanced Research Projects Agency
1400 Wilson Blvd
Arlington, VA 22209

Other Organizations

Dr. H. McI. Parsons
Human Resources Research Office
300 N. Washington Street
Alexandria, VA 22314

Dr. Jesse Orlansky
Institute for Defense Analyses
400 Army-Navy Drive
Arlington, VA 22202

Dr. Arthur I. Siegel
Applied Psychological Services, Inc.
404 East Lancaster Street
Wayne, PA 19087

Dr. Robert T. Hennessy
NAS - National Research Council
JH #819
2101 Constitution Ave., N.W.
Washington, DC 20418

Other Organizations

Dr. Timothy Linquist
Department of Computer Science
VPI & SU
Blacksburg, VA 24061

Dr. M. G. Samet
Perceptronics, Inc.
6271 Variel Avenue
Woodland Hills, CA 91364

Dr. Robert Williges
Human Factors Laboratory
Virginia Polytechnical Institute
and State University
130 Whittemore Hall
Blacksburg, VA 24061

Mr. Edward M. Connelly
Performance Measurement
Associates Inc.
410 Pine Street, S.E.
Suite 300
Vienna, VA 22180